APE — A Set of TeX Macros To Format Ada Programs

Sriram Sankar*

1 Introduction

This report describes a set of macros designed for the purpose of formatting *Ada* [Ada83] programs in TeX[1]. These macros were implemented by the author in early 1987, and they have been refined a few times since. They have also been extended for the language extensions of Ada developed by the *Program Analysis and Verification Group* at Stanford University. Some of these language extensions are *Anna* [LvHKO84] and *TSL* [HL85]. These macros exist as one collection in a file which is included into the document using the `\input` command. This file is available on request from the author.

The design of the macros was motivated by the `programexample` environment in *Scribe*. It is for this reason that they have been named *APE*, which is an acronym for "Ada Program-Example". The macros are of two kinds — *global* macros and *local* macros. The global macros are visible throughout the scope in which the `\input` command has been inserted. The most important global macros are `\apebegin`, `\apeend` and `\ap`. `\apebegin` ... `\apeend` defines an environment in which a complete Ada program can be formatted, while `\ap` is a macro with one argument which is used for insertion of program text within ordinary text. The local macros are available for use only within environments defined by `\apebegin` ... `\apeend` and within the program text of arguments to `\ap`.

2 Formatting Ada Programs using APE

The example below shows a means of formatting a **procedure** EXCH. The following text:

```
\apebegin
\Procedure EXCH(X,Y : \In \Out INTEGER) \Is
   T : INTEGER;
   \cm{2.5in}{T is a temporary variable.}
\Begin
   T := X; X := Y; Y := T;
\End EXCH;
\apeend
```

*Department of Computer Science, Stanford University, Stanford, California 94305.
EMail: sankar@score.stanford.edu.
Phone: (415)723-4962.

[1]These macros can easily be rewritten for other languages.

produces the following output:

procedure EXCH(X,Y : **in out** INTEGER) **is**
    T : INTEGER;
    -- *T is a temporary variable.*
begin
    T := X; X := Y; Y := T;
end EXCH;

The text that produced the very first line in this section is shown below:

```
The example below shows a means of
formatting a \ap{\Procedure EXCH}.
The following text:
```

As is obvious from the above example, keywords are entered as macros whose names are the same as the keywords (with the first letter converted to upper-case). Apart from `\apebegin`, `\apeend` and `\ap`, the keyword macros are the only global macros. Also, comments are inserted using the macro `\cm` which takes two arguments — the first is the maximum width allowed for the comment, and the second is the actual comment itself. If the comment does not fit in one line, it is split into lines each of the specified width. Each of the lines is prefixed by the Ada comment symbol -- in the output[2]. If only the comment symbol is desired (with no comment following it), the macro `\-` can be used. Again, as is obvious from the above example, all spaces and carriage-returns are significant everywhere except within the arguments of the macro `\cm`.

The very first thing that one would like to do is to choose their own favorite fonts for the various Ada constructs. To do this, the following macros should be redefined *after* the `\input` command:

`\apekeywordfont`	`\apkeywordfont`
`\apecommentfont`	`\apcommentfont`
`\apebodyfont`	`\apbodyfont`

Each of the above macros defines a font. Those that are of the form `\ape`... define fonts for the `\apebegin` ... `\apeend` environment, while the others define the fonts for the `\ap` macro. In addition, the following macros can be redefined to specify the horizontal space that each space character inserts; the vertical space that each carriage-return inserts; and the margin at the beginning of each line:

`\apehspace`	`\apevspace`
`\aphspace`	`\apelmargin`

[2]For a similar use of this feature see [Des84].

The range delimiter (..) in Ada is obtained using the macro `\..` The meanings of _ (underscore) and `\_` have been interchanged. Hence underscores in Ada identifiers can be entered without a preceding backslash. However a backslash is required when the underscore is used for subscripting. Most other characters and macros have their usual plain TeX meaning. For example, math mode has to be entered for subscripting and superscripting, and to obtain characters like & and # one has to enter `\&` and `\#` respectively. There is one exception — math mode need not be entered to get a small space; one can enter `\,` directly.

Page-breaks are by default disabled within the `\apebegin` ... `\apeend` environment. However, there are macros that can enable or disable page-breaks within this environment. The macro `\bhinge` enables page-breaks, while the macro `\ehinge` disables page-breaks. Both `\bhinge` and `\ehinge` have global effect — even if they occur inside a group, they continue to have effect outside the group. `\bhinge` and `\ehinge` can be used within comment text (the second parameter to `\cm`) also. When used within comments, these macros also insert a space at their location. It is sometimes useful to specify the particular lines at which page-breaks should be enabled. For this there is a macro — `\\`. `\\` has to be entered at the end of a line just before the carriage-return (`<CR>`). Actually, `\\<CR>` is equivalent to `\bhinge<CR>\ehinge`. Note that enabling page-breaks does not mean that TeX will actually move over to a new page at these locations; it only means that TeX can move over to a new page if its page-breaking algorithms decide that this is necessary. Commands that force page-breaks (like `\eject`) will not work within the `\apebegin` ... `\apeend` environment.

There is also a tabbing facility that can be used within the environment `\apebegin` ... `\apeend`. This is quite similar to the tabbing facility of TeX. For this, there are three more commands, the most important of which is the character `&`. This command shifts to the next tab position. If the current text has overshot this position, then it will back up to the tab position. However, if there is no tab position to skip to, then a new tab position is created at the current distance from the left margin. A carriage-return moves back to before the first tab. The command `\kill` at the end of a line inhibits output of the current line, but any new tab positions created in this line remain set. The command `\actabs` clears all the current tab settings. An implicit `\actabs` is performed by `\apebegin`. Tabbing commands cannot be used within comments created using `\cm`. Any scope that begins within the `\apebegin` ... `\apeend` environment must be ended before the end of the line or the next `&`, whichever is earlier.

Just as in the case of TeX, these tabbing commands format text into a box of appropriate width and inserts `\hss` at the end of the text. Hence it is possible to achieve interesting results like centering text between two tab positions (by inserting glue more infinite than `\hss` on both sides of the text).

To conclude, three more examples are shown below. The first example demonstrates multi-line comments; the second example shows the use of tabbing and different fonts; and the third example shows how centering about a column can be achieved.

Example 1:

The following input:

```
\apebegin
\Procedure CLOSE(FILE: \In \Out FILE_TYPE);
\cm{2.5in}{Severs the association
      between the given file and its
      associated external file.  The
      given file is left closed.  The
      exception {\apebodyfont STATUS_ERROR}
      is raised if the given file is
      not open.}
\apeend
```

produces:

procedure CLOSE(FILE: **in out** FILE_TYPE);
-- *Severs the association between the given*
-- *file and its associated external file. The*
-- *given file is left closed. The exception*
-- STATUS_ERROR *is raised if the given file*
-- *is not open.*

Example 2:

The following input:

```
{\def\apekeywordfont{\normalsize\bf}
\def\apebodyfont{\normalsize\it}
\def\apehspace{0.5em}
\apebegin
\If n < r &\Then n := n + 1;
&\Else &\Begin print_totals; n := 0;
&&\End;
\End \If;
\apeend}
```

produces:

```
if n < r then n := n + 1;
        else begin print_totals; n := 0;
            end;
end if;
```

Example 3:

The following input:

```
\apebegin
\Type DAY \Is (&WEDNESDAY&,&\kill
\Type DAY \Is (&\hfill{SUNDAY}\hfill&,
&\hfill{MONDAY}\hfill&,
&\hfill{TUESDAY}\hfill&,
&\hfill{WEDNESDAY}\hfill&,
&\hfill{THURSDAY}\hfill&,
&\hfill{FRIDAY}\hfill&,
&\hfill{SATURDAY}\hfill&&);
\apeend
```

produces:

```
type DAY is (  SUNDAY   ,
               MONDAY   ,
               TUESDAY  ,
              WEDNESDAY,
              THURSDAY  ,
               FRIDAY   ,
              SATURDAY  );
```

3 Conclusions

The *APE* macros have been used extensively by the author and other members of the *Program Analysis and Verification Group* at Stanford. It has been used to format examples in two books and in many papers. No problems with its use have been encountered so far. The author invites comments and suggestions for the improvement of this set of macros. The complete listing of the macros (for Ada without any extensions) is provided in the Appendix with detailed comments explaining the various aspects of the macro.

References

[Ada83] *The Ada Programming Language Reference Manual.* US Department of Defense, US Government Printing Office, February 1983. ANSI/MIL-STD-1815A-1983.

[Des84] Jacques Désarménien. How to run TEX in a French environment: hyphenation, fonts, typography. *TUGboat*, 5(2):91–102, 1984.

[HL85] David P. Helmbold and David C. Luckham. TSL: task sequencing language. In *Ada in Use: Proceedings of the Ada International Conference*, pages 255–274, Cambridge University Press, May 1985.

[LvHKO84] David C. Luckham, Friedrich W. von Henke, Bernd Krieg-Brückner, and Olaf Owe. *Anna—A Language for Annotating Ada Programs.* Technical Report 84-261, Computer Systems Laboratory, Stanford University, July 1984. (Program Analysis and Verification Group Report 24.)

Appendix A The APE Macros

Editor's note: These macros have been reformatted for presentation in two columns. Problems arising in macros re-keyed from the text below should first be referred to the TUG office. The macros (in their original form) are available for anonymous FTP at `Score.Stanford.edu` in the directory `<TEX.TUGBOAT>`.

```
% Copyright 1988 by Sriram Sankar.
%
% GLOBAL DEFINITIONS:
%
% The fonts for the keywords.
\def\apekeywordfont{\large\bf}
\def\apkeywordfont{\normalsize\bf}

% The fonts for comments.
\def\apecommentfont{\normalsize\sl}
\def\apcommentfont{\normalsize\sl}

% The default fonts for everything else.
\def\apebodyfont{\small\rm}
\def\apbodyfont{\small\rm}

% The space generated by the <space> character.
\def\apehspace{0.65em}
\def\aphspace{0.5em}

% The space inserted between each line.
\def\apevspace{0pt}

% The space left at the beginning of each line.
\def\apelmargin{0pt}

% The above ten macro definitions can be
% redefined within the document as many
% times as desired.  The default settings
% above are in terms of LaTeX macros
% (e.g. \large, \small, etc.).  Note: These
% are the only LaTeX dependencies in this
% file.
```

```
%

\catcode`\!=11
\catcode`\^^I=9
% Tabs will now be ignored, so they can be
% used to format the macros below.
%
% The following macros are for formatting
% the Ada keywords.  Note that they accept
% the keyword as an argument in LOWER-CASE.
% To format keywords in other ways
% (e.g. upper-case), one could either change
% the case of these keyword arguments below,
% or define \apekeywordfont in such a way
% that it changes the case of its argument
% appropriately.

\def\!keyword{\apkeywordfont}

\def\Abort{{\!keyword{abort}}}
\def\Abs{{\!keyword{abs}}}
\def\Accept{{\!keyword{accept}}}
\def\Access{{\!keyword{access}}}
\def\All{{\!keyword{all}}}
\def\And{{\!keyword{and}}}
\def\Array{{\!keyword{array}}}
\def\At{{\!keyword{at}}}
\def\Begin{{\!keyword{begin}}}
\def\Body{{\!keyword{body}}}
\def\Case{{\!keyword{case}}}
\def\Constant{{\!keyword{constant}}}
\def\Declare{{\!keyword{declare}}}
\def\Delay{{\!keyword{delay}}}
\def\Delta{{\!keyword{delta}}}
\def\Digits{{\!keyword{digits}}}
\def\Do{{\!keyword{do}}}
\def\Else{{\!keyword{else}}}
\def\Elsif{{\!keyword{elsif}}}
\def\End{{\!keyword{end}}}
\def\Entry{{\!keyword{entry}}}
\def\Exception{{\!keyword{exception}}}
\def\Exit{{\!keyword{exit}}}
\def\For{{\!keyword{for}}}
\def\Function{{\!keyword{function}}}
\def\Generic{{\!keyword{generic}}}
\def\Goto{{\!keyword{goto}}}
\def\If{{\!keyword{if}}}
\def\In{{\!keyword{in}}}
\def\Is{{\!keyword{is}}}
\def\Limited{{\!keyword{limited}}}
\def\Loop{{\!keyword{loop}}}
\def\Mod{{\!keyword{mod}}}
\def\New{{\!keyword{new}}}
\def\Not{{\!keyword{not}}}
\def\Null{{\!keyword{null}}}
\def\Of{{\!keyword{of}}}
\def\Or{{\!keyword{or}}}
\def\Others{{\!keyword{others}}}
\def\Out{{\!keyword{out}}}
\def\Package{{\!keyword{package}}}
\def\Pragma{{\!keyword{pragma}}}
\def\Private{{\!keyword{private}}}
\def\Procedure{{\!keyword{procedure}}}
\def\Raise{{\!keyword{raise}}}
\def\Range{{\!keyword{range}}}
\def\Record{{\!keyword{record}}}
\def\Rem{{\!keyword{rem}}}
\def\Renames{{\!keyword{renames}}}
\def\Return{{\!keyword{return}}}
\def\Reverse{{\!keyword{reverse}}}
\def\Select{{\!keyword{select}}}
\def\Separate{{\!keyword{separate}}}
\def\Subtype{{\!keyword{subtype}}}
\def\Task{{\!keyword{task}}}
\def\Terminate{{\!keyword{terminate}}}
\def\Then{{\!keyword{then}}}
\def\Type{{\!keyword{type}}}
\def\Use{{\!keyword{use}}}
\def\When{{\!keyword{when}}}
\def\While{{\!keyword{while}}}
\def\With{{\!keyword{with}}}
\def\Xor{{\!keyword{xor}}}
%
% Some more global definitions follow:
%
\def\!lbr{\/$($}
\def\!rbr{\/$)$}
\def\!str{\/$*$}
\def\!pls{\/$+$}
\def\!min{\/$-$}
\def\!col{\/$:$}
\def\!scl{\/$;$}
\def\!les{\/$<$}
\def\!gre{\/$>$}
\def\!bar{\/$|$}
\def\!equ{\/$=$}
%
\def\!sla{\/{\sl{/}}}
\def\!dqt{{\/\raise.2ex\hbox{\tt{"}}}}
\def\!dots{\/.{\hskip1mm}.}
\def\!comment{%
  {\apecommentfont\rm--\thinspace--}}
\def\!hinge{\vfil\penalty5000\vfilneg}
\def\!space{\hskip\apehspace}
\def\!underscore{\underbar{\!space}}
%
% Now follow the global definitions for the
% tabbing commands.  First a set of dimen,
% count and box registers are allocated.
% The number of dimen registers limit the
% total number of tabs permitted.
%
\newdimen\!apetabi
\newdimen\!apetabii
\newdimen\!apetabiii
\newdimen\!apetabiv
\newdimen\!apetabv
\newdimen\!apetabvi
```

```
\newdimen\!apetabvii
\newdimen\!apetabviii
\newdimen\!apetabix
\newdimen\!apetabx
\newdimen\!apetabxi
\newdimen\!apetabxii
\newdimen\!apetabxiii
\newdimen\!apetabxiv
\newdimen\!apetabxv
\newdimen\!apetabxvi
\newdimen\!apetabxvii
\newdimen\!apetabxviii
\newdimen\!apetabxix
\newdimen\!apetabxx
\newcount\!apetotaltabs
\newcount\!apetab
\newbox\!apetabbox
\newbox\!apelinebox
%
% Dimen register i stores the distance from
% the tab stop (i-1) to the tab stop i.
% \!apetotaltabs maintains the total number
% of tabs currently set.  \!apetab maintains
% the tab stop that will be reached if
% another '&' is encountered.  The box
% registers are explained below.
%
% Between the beginning of lines, tab stops
% and the end of lines, text is formatted
% into the box \!apetabbox and padded on
% the right with the glue \hss.  The macro
% \!apebbox is what needs to be done at the
% beginning of each of these boxes.
%
\def\!apebbox{%
  \ifnum\!apetab>\!apetotaltabs
    \setbox\!apetabbox=\hbox
  \else
    \setbox\!apetabbox=\hbox to
      \csname !apetab\romannumeral\!apetab
      \endcsname
  \fi
  \bgroup
  }
%
% The boxes mentioned above (that are stored
% into \!apetabbox) are put together into
% another \hbox called \!apelinebox.  This
% box is output once a full line has been
% read from input.  The macro \!apebline
% does what is needed at the beginning of
% each input line, while the macro \!apeeline
% does what is needed at the end of each
% input line.
%
\def\!apebline{%
  \noindent
  \global\!apetab=1
  \setbox\!apelinebox=
    \hbox\bgroup
      \hskip\apelmargin\!apebbox
  }
%
\def\!apeeline{%
    \hss\egroup
    \box\!apetabbox
  \egroup
  \box\!apelinebox
  }
%
% The macro \!apekill is similar to
% \!apeeline, except that it does not output
% the contents of \!apelinebox.  In addition,
% this macro also performs a \!apebline, thus
% getting ready for the next line of input.
%
\def\!apekill{%
    \hss\egroup
    \box\!apetabbox
  \egroup
  \!apebline
  }
%
% The macro \!apetabskip is expanded when an
% '&' is encountered in the input file.  It
% ends the current \!apetabbox, and if a new
% tab stop needs to be set, it increments the
% counters appropriately and sets the
% appropriate dimen register to the width of
% \!apetabbox.  Finally, \!apebbox is invoked
% to start off a new \!apetabbox.
%
\def\!apetabskip{%
  \hss\egroup
  \ifnum\!apetab>\!apetotaltabs
    \global\advance\!apetotaltabs by 1
    \global\csname
             !apetab\romannumeral\!apetotaltabs
          \endcsname=\wd\!apetabbox
  \fi
  \box\!apetabbox
  \global\advance\!apetab by 1
  \!apebbox
  }
%
\def\!apecr{%
  \strut\par
  \!break
  \vskip\apevspace
  }
%
\def\!par{\!apeeline\!apecr\!apebline}
%
% The following global definitions are for
% formatting Ada comments.  The macros \!cma
% and \!cmb are used in the definition of the
% \cm macro later.  The comment text is split
% at all \bhinge's and \ehinge's.  Each
% portion of the comment is then put into a
% \vbox separately and output one after the
% other.  Actually all but the last line of
% each \vbox is output.  The last line is
% included at the beginning of the next \vbox.
% At the end of it all, there will be one more
% line to output.  \!cmb explicitly outputs
% this line, and moves back to the beginning
% of this line.  The box registers below are
% used to store the boxes created during this
% process.  Dimen register \!cmtpos contains
% the distance of the comment from the left
% margin while \!cmtwd contains the width of
% the comment.
%
\newif\ifnonvoid
\newbox\!cmti
\newbox\!cmtii
\newbox\!cmtiii
\newbox\!cmtiv
\newbox\!cmtv
\newbox\!cmtvi
\newdimen\!cmtpos
\newdimen\!cmtwd
%
% \!cmtout is the output routine.  It assumes
% that the \hboxes to be output are enclosed
% within a \vbox in \!cmti.  It consists
% mainly of two loops.  The first loop
% reverses the order of the \hboxes in \!cmti
% and puts this into a \vbox in \!cmtv.  The
% second loop pulls out the \hboxes from
% \!cmtv and outputs them.
%
\def\!cmtout{%
  \setbox\!cmtii=
    \vbox{
      \unvbox\!cmti
      \global\setbox\!cmtiii=\lastbox
      \unskip
      \unpenalty
      }%
  \nonvoidtrue
  \ifvoid\!cmtiii\nonvoidfalse
  \fi
  \ifnonvoid
    \setbox\!cmti=\box\!cmtii
    \setbox\!cmtv=\vbox{\box\!cmtiii}%
    \nonvoidtrue
    \loop
      \setbox\!cmtii=
        \vbox{
          \unvbox\!cmti
          \global\setbox\!cmtiii=\lastbox
          \unskip
          \unpenalty
          }%
      \ifvoid\!cmtiii\nonvoidfalse
      \fi
    \ifnonvoid
      \setbox\!cmti=\box\!cmtii
      \setbox\!cmtiv=
        \vbox{\unvbox\!cmtv\box\!cmtiii}%
      \setbox\!cmtv=\box\!cmtiv
    \repeat
    \nonvoidtrue
    \loop
      \setbox\!cmtii=
        \vbox{
          \unvbox\!cmtv
          \global\setbox\!cmtiii=\lastbox
          }%
      \ifvoid\!cmtiii\nonvoidfalse
      \fi
    \ifnonvoid
      \setbox\!cmtv=\box\!cmtii
      \noindent\hskip\!cmtpos
      \!comment{}\!space
      \box\!cmtiii{}\!apecr
    \repeat
  \fi
  }
%
% The following two macros \!cmtbhinge and
% \!cmtehinge end the current comment box
% being created.  They then pull out the
% last line from this box and ship the
% rest of the box to \!cmtout.  The value
% of \!break is then modified to ensure that
% the correct penalty value is inserted at
% the end of the next line.  They then start
% off a new comment box after inserting the
% last line of the previous box and a space
% character at the beginning.
%
\def\!cmtbhinge{%
  \egroup
  \setbox\!cmtii=
    \vbox{
      \unvbox\!cmti
      \global\setbox\!cmtiii=\lastbox
      \unskip
      \unpenalty
      }%
  \setbox\!cmtvi=
    \hbox{%
      \unhbox\!cmtiii
      \unskip\unskip
      \unpenalty
      }%
  \setbox\!cmti=\box\!cmtii
  \!cmtout
  \global\def\!break{\!hinge}%
  \setbox\!cmti=
    \vbox\bgroup
      \hsize=\!cmtwd
      \noindent\apecommentfont
      {}\unhbox\!cmtvi{} %
  }
%
\def\!cmtehinge{%
  \egroup
  \setbox\!cmtii=
    \vbox{
      \unvbox\!cmti
      \global\setbox\!cmtiii=\lastbox
      \unskip\unpenalty
      }%
  \setbox\!cmtvi=
    \hbox{%
      \unhbox\!cmtiii
      \unskip\unskip
      \unpenalty
      }%
  \setbox\!cmti=\box\!cmtii
  \!cmtout
  \global\def\!break{\nobreak}%
  \setbox\!cmti=
    \vbox\bgroup
      \hsize=\!cmtwd
      \noindent\apecommentfont
      {}\unhbox\!cmtvi{} %
  }
%
% \!cma begins the comment by finishing off
% the boxes being created and then outputting
% them.  It also measures the width of the
% box output to determine the value of
% \!cmtpos.
%
\def\!cma{%
    \egroup
    \setbox\!cmti=\hbox{\unhbox\!apetabbox}%
    \box\!cmti
  \egroup
  \!cmtpos=\wd\!apelinebox
  \hbox to 0pt{\box\!apelinebox\hss}%
}
%
% \!cmb takes over from \cm.  It starts off
% the first comment box and finishes
% off the last comment box.  If there is
% only one comment box (no \bhinge or
% \ehinge in between), then it starts and
% finishes this one box.  It then pulls
% out the last line of the last box and
% ships the rest of the box to \!cmtout.
% It then outputs the last comment line
% and moves to the beginning of the line
% in which the last comment line was output.
%
\def\!cmb#1#2{%
    \!cmtwd=#1%
    \setbox\!cmti=
      \vbox\bgroup
        \hsize=\!cmtwd
        \noindent
        \apecommentfont
        {}#2\egroup
    \setbox\!cmtii=
      \vbox{
        \unvbox\!cmti
        \global\setbox\!cmtvi=\lastbox
        \unskip
        \unpenalty
        }%
    \setbox\!cmti=\box\!cmtii
    \!cmtout
    \noindent
    \hbox to 0pt{%
      \hskip\!cmtpos
      \!comment{}\!space
      \box\!cmtvi
      \hss
      }%
  \endgroup
  \!apebline
  }
%
% The following global definition of \!ap is
% used in the definition of the \ap macro
% later.  As in the case of \!cmb and \cm,
% there is an \endgroup in \!ap whose
% corresponding \begingroup is in \ap.  In
% this case, there is also a corresponding
% \apebegin in \ap.
%
\def\!ap#1{#1\apeend\endgroup{}}
%
% We cannot have the sequence "\let=\!equ"
% within the body of the ape macro since
% the "=" will not be scanned with the
% correct category code.  Hence, the macro
% \!defequ is defined in an environment
% where "=" has the correct category code,
% and then this macro is used within the body
% of the ape macro.  \!defequ is now defined
% as a global macro.
%
{
\catcode`\==\active
\gdef\!defequ{\let=\!equ}%
}
%
\catcode`\!=12
\catcode`\^^I=10
%
% END OF GLOBAL DEFINITIONS.
%
% NOW THE MACRO DEFINITIONS:
%
% The macro has to be defined in an
% environment with the category codes set
% correctly.  This environment is first set
% up below, and then within this environment,
% the macros are defined.
```

```
%
{% The macro definition environment
%
% Firstly, "outer" macros cannot occur
% within any macro expansions.  The only
% such macro that occurs in the macros
% below is \+.  Hence this is redefined to
% its usual value.  The effect of this
% redefinition just changes the status of
% \+ to now be a non-"outer" macro.
%
\def\+{\tabalign}%
%
% Necessary changes to category codes are
% made right now since category codes
% are used while scanning and the macro
% is scanned at definition time, and not
% at macro expansion time.
%
\catcode`\(=\active%
\catcode`\)=\active%
\catcode`\*=\active%
\catcode`\+=\active%
\catcode`\-=\active%
\catcode`\:=\active%
\catcode`\;=\active%
\catcode`\<=\active%
\catcode`\>=\active%
\catcode`\|=\active%
\catcode`\/=\active%
\catcode`\"=\active%
\catcode`\^^M=\active%
\catcode`\ =\active%
\catcode`\_=\active%
\catcode`\!=11%
\catcode`\&=\active%
%
\gdef\apebegin{%
\begingroup%
\global\def\!break{\nobreak}%
%
% First the category codes of the characters
% to be redefined are made active.
%
\catcode`\(=\active%
\catcode`\)=\active%
\catcode`\*=\active%
\catcode`\+=\active%
\catcode`\-=\active%
\catcode`\:=\active%
\catcode`\;=\active%
\catcode`\<=\active%
\catcode`\>=\active%
\catcode`\|=\active%
\catcode`\/=\active%
\catcode`\"=\active%
\catcode`\^^M=\active%
\catcode`\ =\active%
\catcode`\_=\active%
\catcode`\!=11%
\catcode`\&=\active%
%
% Now each of these characters is given
% its definition.
%
\let(=\!lbr%
\let)=\!rbr%
\let*=\!str%
\let+=\!pls%
\let-=\!min%
\let:=\!col%
\let;=\!scl%
\let<=\!les%
\let>=\!gre%
\let|=\!bar%
\let/=\!sla%
\let"=\!dqt%
\let^^M=\!par%
\let =\!space%
\let_=\!underscore%
\catcode`\^^A=8\def\_{^^A}%
%
% The above two lines interchange the
% meanings of _ and \_.  To do this, a new
% subscript character ^A is defined, and
% then \_ is defined as a macro to
% expand to ^A.
%
\let&=\!apetabskip%
%
% Now follows the remaining macro
% declarations needed to complete the \ape
% environment.
%
\def\.{\!dots}%
\def\,{\thinspace{}}%
\def\-{\!comment}%
%
\def\bhinge{\global\def\!break{\!hinge}}%
\def\ehinge{\global\def\!break{\nobreak}}%
\def\\^^M{\bhinge^^M\ehinge}%
\def\actabs{%
\global\!apetotaltabs=0%
\global\!apetab=1{}}% end \def of actabs
\actabs%
\def\kill^^M{\!apekill}%
\def\!keyword{\apekeywordfont}%
%
% Following is the \cm macro definition.
% This macro eventually lets the macro
% \!cmb take over so that the text
% following is read in with the correct
% category codes.  The redefinition of
% \+ below is for the same reason as
% before: to convert it from an outer
% to a non-outer macro.
%
\def\+{\tabalign}%
%
\def\cm{\!cma\begingroup%
%
% The category codes are temporarily
% changed back to their original values.
%
\catcode`\(=12%
\catcode`\)=12%
\catcode`\*=12%
\catcode`\+=12%
\catcode`\-=12%
\catcode`\:=12%
\catcode`\;=12%
\catcode`\<=12%
\catcode`\>=12%
\catcode`\|=12%
\catcode`\/=12%
\catcode`\"=12%
\catcode`\^^M=5%
\catcode`\ =10%
\catcode`\!=12%
%
\def\bhinge{\!cmtbhinge}%
\def\ehinge{\!cmtehinge}%
%
\!cmb}%
%
% Now = is redefined.  Had this been done
% earlier, then it would have affected
% the previous definitions, since many
% of them contain =.
%
\catcode`\==\active\!defequ%
%
\apebodyfont\!apebline%
%
}% end of definition of \apebegin
%
\gdef\apeend{\!apeeline\endgroup}%
%
\gdef\ap{\begingroup%
\def\apekeywordfont{\apkeywordfont}%
\def\apecommentfont{\apcommentfont}%
\def\apebodyfont{\apbodyfont}%
\def\apehspace{\aphspace}%
\def\apelmargin{0pt}%
\apebegin\!ap%
}%
%
% The reason for letting \!ap take over
% from \ap at this point is to make TeX
% scan the argument with the correct
% category codes.
%
}% This ends the environment in which the
% macros are defined.
%
```

LaTeX

Contents of Archive Server as of 16 January 1989

Michael DeCorte
Clarkson University

Several changes to the LaTeX Archive have been made since the last TUGboat. It has been split into several different subarchives: LaTeX style files; TeX style files; AMS-TeX style files; BibTeX style files; BibTeX 0.98 style files; TeX programs. Also there are several new archives: AMS-TeX source; BibTeX source; CM fonts source; LaTeX source; TeX documentation; TeX inputs; TeX source; TeX tests; TUGboat files; TeXhax digests; TeXMaG digests; UKTeX digests.

As always, submissions are encouraged. If you do submit a file please include at the top of the file: your name; your email address; your real address; the date. Also please make certain that there are no lines in the file longer than 80 characters as some mailers will truncate them. Mail should be sent to

`mrd@sun.soe.clarkson.edu`
`archive-management@sun.soe.clarkson.edu`

For Internet users: How to ftp

An example session is shown at the bottom of the following page. Users should realize that ftp syntax varies from host to host. Your syntax may be different. The syntax presented here is that of Unix ftp. Comments are in parentheses. The exact example is for retrieving files from the LaTeX Archive; the syntax is similar for the other archives, only the directories differ. The directory for each archive is given in its description.

Non-Internet users: How to retrieve by mail

To retrieve files or help documentation, send mail to `archive-server@sun.soe.clarkson.edu` with the body of the mail message containing the command `help` or `index` or `send`. The `send` command must be followed by the name of the archive and then the files you want. Users who are not in the uucp maps database are strongly encourage to include a `path` command followed by a path from Clarkson to you in domain style format. If you don't include a `path` command, your mail may not get to you and will definitely be delayed as Michael will have to mail it by hand. You should realize that Clarkson does not have a uucp connection; therefore you must send it to an Internet or Bitnet